LLNL-CONF-405321

# Designing Pu600 for Authentication

G. White

July 11, 2008

Institute of Nuclear Materials Management
Nashville, TN, United States
July 13, 2008 through July 17, 2008

# Designing Pu600 for Authentication

Greg White
Lawrence Livermore National Laboratory
July 2008

**Abstract**
Many recent Non-proliferation and Arms Control software projects include an authentication component. Demonstrating assurance that software and hardware performs as expected without hidden "back-doors" is crucial to a project's success. In this context, "authentication" is defined as determining that the system performs only its intended purpose and performs that purpose correctly and reliably over many years. Pu600 is a mature software solution for determining the presence of Pu and the ratio of Pu240 to Pu239 by analyzing the gamma ray spectra in the 600 KeV region. The project's goals are to explore hardware and software technologies which can by applied to Pu600 which ease the authentication of a complete, end-to-end solution. We will discuss alternatives and give the current status of our work.

## 1 - Introduction to Authentication

As we make progress toward the deployment of monitoring systems for nuclear material, two important goals must be observed: protection of the host country's sensitive information and assurance to the monitoring party that the nuclear material is what the host country has declared it to be. These goals are met by *certification* in the host country and *authentication* by the monitoring party. During both certification and authentication, each side needs to understand all of the operating parameters of the hardware and software in the deployed system. This paper concentrates on software authentication, but similar principles apply to hardware authentication, as well as to software and hardware certification.

Authentication is the process of gaining assurance that a system is performing robustly and precisely as intended. The simpler the system, the easier it is to authenticate. It is important to limit functionality to only what is needed to satisfy the requirements of the task. Each design decision makes authentication easier, or harder. For example, a design with Microsoft MS-DOS (which requires a 4.77 MHz processor and runs on a single 1.44 MB floppy disk) is significantly easier to authenticate than a Windows Vista installation (which requires an 1 GHz processor 512 MB of memory, and 15 GB of free disk space).[1] Simpler hardware, expressed in the number of gates, chips, or boards, is easier to authenticate than more complex hardware. The same can be said for application and development software.

Other industries have a similar need for authentication. Computers that perform electronic voting[2] and gambling are disparate examples. In previous INMM papers,[3,4,5,6] we have discussed a hypothetical perfect system for authentication, with transparent (to both parties) hardware and software development, and advocated "open source" hardware and software solutions. We advocated software language choices that lower authentication costs, specifically comparing procedural languages with object-oriented languages. In particular, we examined the C and C++ languages, comparing language features, code generation, implementation details, and executable image size, and demonstrated how these attributes aid or hinder authentication. We showed that programs in lower level, procedural languages are more easily authenticated than object-oriented ones. We suggested some possible ways to mitigate the use of object-oriented programming languages. We described the scope of the software authentication process and the five methods of software authentication. We then concentrated on different types of source code analysis,

LLNL-CONF-405321

introducing LLNL's ROSE software tool for automating the authentication of source code. Finally, we discussed how authentication of binaries is complementary to source code authentication.

## 2 – Pu600

Pu600[7,8] is widely recognized as the preferred method for determining the presence of plutonium and the Pu240/Pu239 ratio of a sample material using gamma rays. It has a long history of successful deployments under a wide range of measurement regimes. Pu600 shares a code base with MGA[9], which has a multiple decade lifetime and is available commercially through ORTEC. Pu600 currently runs under MSDOS on PC-compatible platforms. It is written in FORTRAN and requires approximately a 100 MHz Intel 80486DX processor to obtain quick answers.

## 3 – Project Goals

The project's goals are to use Pu600 as a testbed for evaluating technologies and methods to ease authentication. Pu600 is representative of the kinds of physics codes applied to these types of problems. Our end-state is a small, simple, single board computer with two or three serial ports. One serial port will connect to a data acquisition system and the HPGe gamma ray detector. The second serial port will connect to the computational block, which receives the results of Pu600. An optional third serial port could be used for debugging or to show intermediate results when it is taking data in the open mode. Read-Only Memory (ROM) will hold the software, and volatile Random Access Memory (RAM) will hold the data and intermediate results. The hardware should not include additional features or functionality and should be fully documented.

The software language choice is critical to the project. To increase portability, we chose to translate the Pu600 application from FORTRAN to C. FORTRAN compilers tend to be commercial, closed source, expensive, and not available on all hardware and operating systems. C compilers, in general, are available in commercial and open source, less expensive, and available for almost all hardware and operating systems. The C language is procedural and a good choice for authentication.[10] It is also critical that Pu600 retain its dependability, readability, and maintainability after conversion to the C language. Also, other parts of the system will probably be written in C, so the conversion of Pu600 will decrease the number of compilers needed in the development system from two to one.

## 4 – Current Status

There are two general approaches to translating from one computer language to another: manual conversion, or automated conversion. Manual conversion is tedious and error prone. Historically, automatic language conversions tended to produce correct, but unreadable and unmaintainable code. An example of this automatic language conversion is the *f2c* project.[11] In fact, many projects which utilize this approach continue to develop the original code in FORTRAN, only using the *f2c* right before giving it to the C compiler to create an executable. This method adds unneeded complexity to the authentication process, since it adds the *f2c* software to the authentication process. Applications converted with *f2c* run approximately twice as slow as the original FORTRAN executable. Another option is *ForPasC*[12], but it is written for MSDOS and has not been actively developed since 1991.

Instead, we found a commercial source-to-source translator from FORTRAN to C that met our requirements. *FOR_C*[13] by Cobalt Blue produces readable and maintainable code. The FORTRAN and C versions have similar execution times.[14] Conversion options were chosen to maximize readability and minimize code size. Also, we let *FOR_C* automatically convert input/output from FORTRAN *read, write,* and *format* statements to *fprintf()* and *fscanf()* statements. We chose not to utilize the *FOR_C* runtime library, since we didn't really need it, and not using it decreased the amount of source code for Pu600.

We have created an automated build and test suite using a collection of sample gamma ray spectra provided by the original developer of Pu600. The C and FORTRAN versions have been ported to both Linux and MacOS X using the GNU Compiler Collection. More importantly, from the beginning, all versions of the code produces the same (character for character) primary output and results as the original MSDOS FORTRAN executable. This is a critical result with a huge cost savings over a manual conversion process. We continuously maintain backward equality with the original, trusted version of the code.

We continue to work to increase the readability of the code. One place where *FOR_C* didn't produce perfect results was in keeping comments with the appropriate source code in all cases. It was a trivial matter to manually move comments to match their original position. Like similar codes originally written in FORTRAN decades ago, Pu600 inherited from MGA a large number of GOTO statements. As Cobalt Blue states on their webpage, "…spaghetti FORTRAN will, unfortunately, result in spaghetti C". We are working through the code, manually converting GOTO statements to loops, branches, etc. In fact, we have reduced the number of GOTO statements by almost half. *FOR_C* also keeps two copies of loop variables, one that uses the C convention for array indexing (start at zero), and one that uses the FORTRAN convention for array indexing (start at one). Where possible, we removed unneeded loop variables, refactoring code as necessary. We continue to refine and improve the similarity of intermediate results between the C and FORTRAN versions of Pu600, to increase confidence in the integrity of the conversion.

To evaluate the software quality of Pu600 (and the conversion process to C) we have collaborated with LLNL's Software Quality Assurance Group. We utilized the *Gcov*[15] tool from the GNU Compiler Collection to assess code coverage. The code coverage result was 68%, which is comparable with the highest quality, large physics codes used in the DOE ASCI Program. We also assessed the software quality of the C version of Pu600 using the commercial source code static analysis tool by KlocWork[16].

## 5 – Next Steps

We continue to research suitable hardware and operating system platforms for running the code. We have been unable to find a commercial hardware platform that meets the needs of authentication and data protection. Most commercial hardware platforms use large quantities of writable flash memory, which we cannot use. We are pursuing a collaboration with the Electrical Engineering Department of UC Davis to produce a custom, open hardware platform to run Pu600 on. Hardware will be tailored to exactly match the needs of the project. We believe this option could decrease development costs.

Work on porting and demonstrating Pu600 on an open source embedded operating system will commence shortly. It is important that we continue to use an open source software development tool chain (such as the GNU Compiler Collection). While we resolve the aforementioned hardware issues, we will demonstrate Pu600 on interim commercial hardware. We expect to use the eCos[17] embedded operating system. eCos has been used in Brother laser printers and the Iomega HipZip MP3 Player, which share similarities with our needs. eCos supports a wide range of processors, which allow us additional flexibility. One of eCos's strengths is that it offers developers the ability to easily remove portions of the operating system not needed for a specific application. Over 200 configuration options can be set. This allows developers to minimize the amount of software used in the system, which will aid authentication and certification. The eCos operating system can be paired down to occupy only four kilobytes of memory. We will also explore tradeoffs with running Pu600 without an operating system.

*This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.*

## References

[1]http://www.microsoft.com/windows/products/windowsvista/editions/systemrequirements.mspx
[2] As an aside, a genius co-worker of mine stated, “If I wanted to rig an election with an electronic voting machine, and if I could choose any computer language to write in to hide my deception in, I’d do it in C++.”
[3] White, G., Increasing Inspectability of Hardware and Software for Arms Control and Nonproliferation Regimes, *Proceedings of the INMM 2001 Annual Meeting*, Indian Wells, California
[4] White, G., Computer Language Choices in Arms Control and Nonproliferation Regimes, *Proceedings of the INMM 2005 Annual Meeting*, Phoenix, Arizona
[5] White, G., Strengthening Software Authentication with the ROSE Software Suite, *Proceedings of the INMM 2006 Annual Meeting*, Nashville, Tennessee
[6] White, G., Tools and Methods for Increasing Trust in Software, *Proceedings of the INMM 2007 Annual Meeting*, Tuscon, Arizona
[7] Luke, S. J., White, G. K., Archer, D. E., Wolford, J. K., Gosnell, T.B., Verification of the Presense of Weapons-Quality Plutonium in Sealed Storage Containers for the Trilateral Initiative Demonstration, *Symposium International Safeguards: Verification and Nuclear Material Security,* Vienna, Austria, October 29-November 1, 2001. UCRL-JC-145918, https://e-reports-ext.llnl.gov/pdf/246934.pdf
[8] Luke, S. John and Archer, Daniel E., Gamma Attribute Measurements – Pu300, Pu600, Pu900, *41st Annual Meeting of the Institute of Nuclear Materials Management, New Orleans, LA, July 15-20, 2000,* https://e-reports-ext.llnl.gov/pdf/238167.pdf
[9] R. Gunnink, MGA: A Gamma-Ray Spectrum Analysis Code for Determining Plutonium Isotopic Abundances, Lawrence Livermore National Laboratory, Livermore, CA, UCRL-LR-103220, April 1990, Vol. 1-2.
[10] White, G., Computer Language Choices in Arms Control and Nonproliferation Regimes, *Proceedings of the INMM 2005 Annual Meeting*, Phoenix, Arizona
[11] http://www.netlib.org/f2c/
[12] http://irpcsoft.com/
[13] http://www.cobalt-blue.com
[14] http://cobalt-blue.com/fc/fcfaqs.htm “With most code, there isn't much difference. Some programs run faster in C (usually due to faster C compilers), and some run faster in FORTRAN. Integer arithmetic and native I/O is normally faster in C, whereas complex arithmetic is definitely faster in FORTRAN. Double precision arithmetic is about the same in both languages.”
[15] http://gcc.gnu.org/onlinedocs/gcc/Gcov.html
[16] http://www.klocwork.com/
[17] http://www.ecoscentric.com/